Algorithmica (2011) 61:674–693
DOI 10.1007/s00453-010-9430-0

Preprocessing Imprecise Points for Delaunay Triangulation: Simplified and Extended

Kevin Buchin · Maarten Löffler · Pat Morin · Wolfgang Mulzer

Received: 2 September 2009 / Accepted: 5 July 2010 / Published online: 28 July 2010

Abstract Suppose we want to compute the Delaunay triangulation of a set P whose points are restricted to a collection $\mathcal{R}$ of input regions known in advance. Building on recent work by Löffler and Snoeyink, we show how to leverage our knowledge of $\mathcal{R}$ for faster Delaunay computation. Our approach needs no fancy machinery and optimally handles a wide variety of inputs, e.g., overlapping disks of different sizes and fat regions.

Keywords Delaunay triangulation · Data imprecision · Quadtree

This research was partially supported by the Netherlands Organisation for Scientific Research (NWO) through the project GOGO, the BRICKS/FOCUS project no. 642.065.503, and project no. 639.022.707. P. Morin was supported by NSERC, CFI, and the Ontario Ministry of Research and Innovation. W. Mulzer was supported in part by a Wallace Memorial Fellowship in Engineering and NSF grant CCF-0634958 and NSF CCF 0832797.

K. Buchin
Dept. of Mathematics and Computer Science, TU Eindhoven, Eindhoven, The Netherlands
e-mail: kbuchin@win.tue.nl

M. Löffler
Computer Science Department, University of California, Irvine, CA 92697, USA
e-mail: mloffler@uci.edu

P. Morin
School of Computer Science, Carleton University, Ottawa, Canada
e-mail: morin@scs.carleton.ca

W. Mulzer (✉)
Department of Computer Science, Princeton University, Princeton, NJ 08540, USA
e-mail: wmulzer@cs.princeton.edu

1 Introduction

Data imprecision is a fact of life that is often ignored in the design of geometric algorithms. The input for a typical computational geometry problem is a finite point set P in $\mathbb{R}^2$, or more generally $\mathbb{R}^d$. Traditionally, one assumes that P is known exactly, and indeed, in the 1980s and 1990s this was often justified, as much of the input data was hand-constructed for computer graphics or simulations. Nowadays, however, the input is often sensed from the real world, and thus inherently imprecise. This leads to a growing need to deal with imprecision.

An early model for imprecise geometric data, motivated by finite precision of coordinates, is called *ε-geometry* [22]. Here, the input is a traditional point set P and a parameter ε. The true point set is unknown, but each point is guaranteed to lie in a disk of radius ε. Even though this model has proven fruitful and remains popular due to its simplicity [2, 23], it may often be too restrictive: imprecision regions could be more complicated than disks, and their shapes and sizes may even differ from point to point, e.g., to model imprecision from different sources, independent imprecision in different input dimensions, etc. In these settings, the extra freedom in modeling leads to more involved algorithms, but still many results are available [26, 28, 30, 31].

1.1 Preprocessing

The above results assume that the imprecise input is given once and simply has to be dealt with. While this holds in many applications, it is also often possible to get (more) precise estimates of the points, but they will only become available later when there is less time, or they come at a higher cost. For example, in the *update complexity* model [10, 21], each data point is given imprecisely at the beginning but can always be found precisely at a certain price.

One model that has received attention lately is that of *preprocessing* an imprecise point set so that some structure can be computed faster when the exact points become available later. Here, we consider triangulations: let $\mathcal{R}$ be a collection of n planar regions, and suppose we know that the input has exactly one point from each region. The question is whether we can exploit our knowledge of $\mathcal{R}$ to quickly triangulate the exact input, once it is known. More precisely, we want to preprocess $\mathcal{R}$ into a data structure for the following problem: given a point p_i from each region $R_i \in \mathcal{R}$, compute a triangulation of $P = \{p_1, \ldots, p_n\}$.[1] For reasons that will become clear below, we call this data structure a *scaffolding*, and we will refer to the process of finding a triangulation for P as *collapsing* the scaffolding for a concrete point set P. There are many parameters to consider; not only do we want the resources required to build, store, and collapse the scaffolding to be small, but we would also like to support general classes of input regions and obtain "nice" (i.e., Delaunay) triangulations. In the latter case, we say that we collapse the scaffolding into a Delaunay triangulation for a concrete point set P.

[1] We will assume that all inputs are valid, i.e., that p_i is in R_i for all i. The complexity of checking this depends on the complexity of and the representation of the regions $R_1, \ldots, R_n$.

Held and Mitchell [24] show that if $\mathcal{R}$ consists of n disjoint unit disks, it can be preprocessed in $O(n \log n)$ time into a linear-space data structure such that a triangulation for a point set with exactly one point from every disk can be found in linear time. This is improved by Löffler and Snoeyink [29] who show that it is possible to construct a linear-size data structure such that it takes linear time to find the *Delaunay* triangulation for a set with one point from every disk. Both results generalize to regions with limited overlap and limited difference in shape and size—as long as these parameters are bounded by a constant, the same results hold. However, no attempt is made to optimize the dependence on the parameters.

Contrarily, van Kreveld, Löffler, and Mitchell [27] study the case when $\mathcal{R}$ consists of n disjoint polygons with a total of m vertices, and they obtain an $O(m)$-space data structure with $O(m \log m)$ preprocessing time so that a triangulation for a point set with one point in each polygon can be found in linear time. There is no restriction on the shapes and sizes of the individual regions (they do not even strictly have to be polygonal), only on the overlap. As these works already mention, a similar result for *Delaunay* triangulations is impossible. Djidjev and Lingas [18] show that if the points are sorted in any one direction, it still takes $\Omega(n \log n)$ time to compute their Delaunay triangulation. If $\mathcal{R}$ consists of vertical lines, the only information we could precompute is exactly this order (and the distances, but they can be found from the order in linear time anyway). All the algorithms above are deterministic.

1.2 Contribution

Our main concern will be Delaunay triangulations. First, we show that the algorithm by Löffler and Snoeyink [29] can be simplified considerably if we are happy with randomization and expected running time guarantees. In particular, we avoid the need for linear-time polygon triangulation [14], which was the main tool in the previous algorithm.

Second, although there can be no data structure for finding a Delaunay triangulation for points from arbitrary regions in $o(n \log n)$ time, we show that for realistic input we can get a better dependence on the realism parameters than in [29]. In particular, we consider k, the largest depth in the arrangement of $\mathcal{R}$, and β_f, the smallest fatness of any region in $\mathcal{R}$ (defined for a region R as the largest β such that for any disk D with center in R and intersecting ∂R, $\text{area}(R \cap D) \geq \beta \cdot \text{area}(D)$). We can build in $O(n \log n)$ time a scaffolding for $\mathcal{R}$ of linear size that can be collapsed to a Delaunay triangulation in time $O(n \log(k/\beta_f))$. We also consider β_t, the smallest thickness (defined as the fraction of the outcircle of a region occupied by it) of any region in $\mathcal{R}$, and r, the ratio between the diameters of the largest and the smallest region in $\mathcal{R}$. With the same preprocessing time and space, we can obtain a scaffolding that can be collapsed to a Delaunay triangulation in $O(n(\log(k/\beta_t) + \log\log r))$ time. For comparison, the previous bound is $O(nkr^2/\beta_t^2)$ [29]. Finally, we achieve similar results in various other realistic input models.

We describe two different approaches. The first, which gives the same result as [29], is extremely simple and illustrates the general idea. The second approach relies on quadtrees [5, Chap. 14] and is a bit more complicated, but generalizes easily. As hinted above, we use a technique that has emerged just recently in the literature [16, 17, 27] and which we call *scaffolding*: in order to compute many related

structures quickly, we first compute a "typical" structure—the *scaffolding* Q—in a preprocessing phase. To process a given point set, we insert the points into Q and then remove Q. In the first approach, the scaffolding is a Delaunay triangulation for a suitable point set, and we will use an algorithm for *hereditary* Delaunay triangulations [16] to collapse it efficiently. In particular, we need the following result:

Theorem 1.1 (Chazelle et al. [15], see also [16]) *Let $P, Q \subseteq \mathbb{R}^2$ be two planar point sets with $|P \cup Q| = m$, and suppose that $\mathrm{DT}(P \cup Q)$ is available. Then $\mathrm{DT}(P)$ can be computed in expected time $O(m)$.*

In the second approach, the scaffolding is an appropriate quadtree. To collapse it, we employ a recent result that connects quadtrees with Delaunay triangulations

Theorem 1.2 (Buchin and Mulzer [11]) *Let P be a planar n-point set and suppose that a quadtree T for P is available.*[2] *Then $\mathrm{DT}(P)$ can be computed in expected time $O(n)$.*

We will say more about this theorem in Appendix B, where we also prove a slight generalization that is needed by our algorithm.

2 Unit Disks: Simplified Algorithm

We begin with a very simple randomized algorithm for the original setting: given a sequence of n disjoint unit disks $\mathcal{R} = \langle R_1, \ldots, R_n \rangle$, we show how to preprocess $\mathcal{R}$ in $O(n \log n)$ time into a linear-space scaffolding that can be collapsed to a Delaunay triangulation in $O(n)$ *expected* time.

Let c_i denote the center of R_i and for $r > 0$ let R_i^r be the disk centered at c_i with radius r. The preprocessing algorithm creates a point set Q that for each R_i contains c_i and 7 points equally spaced on ∂R_i^2, the boundary of R_i^2. Then it computes $\mathrm{DT}(Q)$, the Delaunay triangulation of Q, and stores it. Since Q has $8n$ points, this takes $O(n \log n)$ time (e.g., [5, Sect. 9]). We will need the following useful lemma about Q.

Lemma 2.1 *Let X be a point set with at most one point from each R_i. Any disk D of radius r contains at most $9(r+3)^2$ points of $Q \cup X$.*

Proof Let c be the center of D. Any point of $Q \cup X$ in D comes from some R_i with $\|c - c_i\| \leq r + 2$. The number of such R_i is at most the number of disjoint unit disks that fit into a disk of radius $r + 3$. A simple volume argument bounds this by $(r+3)^2$. As each R_i contributes up to 9 points, the claim follows. □

Given the sequence $P = \langle p_1, \ldots, p_n \rangle$ of precise points, we construct $\mathrm{DT}(P)$ by first inserting P into $\mathrm{DT}(Q)$ to obtain $\mathrm{DT}(Q \cup P)$ and then applying Theorem 1.1

[2] In Sect. 3, we will define precisely what it means that T is a quadtree for P.

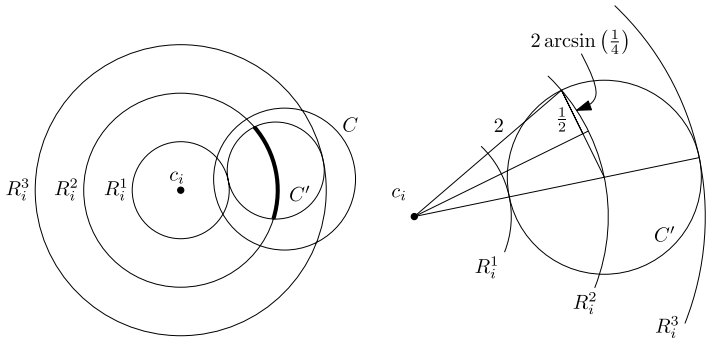

Fig. 1 C' covers a constant fraction of the boundary of R_i^2 and hence meets Q

to remove Q. To compute $\mathrm{DT}(Q \cup P)$, we proceed as follows: for each point p_i we perform a (breadth-first or depth-first) search among the triangles of $\mathrm{DT}(Q \cup \{p_1, \ldots, p_{i-1}\})$ that starts at some triangle incident to c_i and never leaves R_i, until we find the triangle t_i that contains p_i. Then we insert p_i into $\mathrm{DT}(Q \cup \{p_1, \ldots, p_{i-1}\})$ by making it adjacent to the three vertices of t_i and performing *Delaunay flipping* [5, Sect. 9.3]. This takes time proportional to the number of triangles visited plus the degree of p_i in $\mathrm{DT}(Q \cup \{p_1, \ldots, p_i\})$. The next lemma allows us to bound these quantities.

Lemma 2.2 *Let $Y = Q \cup X$ where X is any point set. Let C be a disk whose interior does not meet Y and such that C intersects R_i. Then C lies entirely inside R_i^3.*

Proof We prove the contrapositive. Suppose C is a disk that intersects both R_i and $\mathbb{R}^2 \setminus R_i^3$. Then C contains a (generally smaller) disk C' tangent to R_i^1 and to ∂R_i^3; see Fig. 1. The intersection of C' with ∂R_i^2 is a circular arc of length $8\arcsin(1/4) > 4\pi/7$. But this means that one of the 7 points on ∂R_i^2 must lie inside $C' \subseteq C$, so the interior of C contains a point in Y. □

Lemma 2.2 immediately implies the following:

Lemma 2.3 *Let $Y = Q \cup X$ where X is any point set and consider $\mathrm{DT}(Y)$. Then, for any point $p \in Y \cap R_i$, all neighbors of p in $\mathrm{DT}(Y)$ lie inside R_i^3.*

Proof If pq is an edge of $\mathrm{DT}(Y)$, there exists a circle C that has p and q on its boundary and whose interior does not meet Y. Since $p \in R_i$, Lemma 2.2 implies that C is contained in R_i^3, so $q \in R_i^3$, as claimed. □

The next two lemmas bound the number of triangles visited while inserting p_i.

Lemma 2.4 *Any triangle of $\mathrm{DT}(Q \cup \{p_1, \ldots, p_i\})$ that intersects R_i has all three vertices in R_i^3.*

Proof Let t be a triangle that intersects R_i, and let C be its circumcircle. Since C intersects R_i and has empty interior, Lemma 2.2 implies $t \subseteq C \subseteq R_i^3$, as claimed. □

Lemma 2.5 *At most* 644 *triangles of* $\mathrm{DT}(Q \cup \{p_1, \ldots, p_i\})$ *intersect* R_i.

Proof The triangles that intersect R_i form the of faces of a planar graph G. By Lemma 2.4 every vertex of G lies inside R_i^3, so by Lemma 2.1, there are at most $v = 9(3+3)^2 = 324$ vertices, and thus at most $2v - 4 = 644$ faces. □

The final lemma bounds the degree of p_i at the time it is inserted.

Lemma 2.6 *The degree of* p_i *in* $\mathrm{DT}(Q \cup \{p_1, \ldots, p_i\})$ *is at most* 324.

Proof By Lemma 2.3, all the neighbors of p_i are inside R_i^3 and by Lemma 2.1 there are at most $9(3+3)^2 = 324$ points of $Q \cup \{p_1, \ldots, p_i\}$ in R_i^3. □

Thus, by Lemmas 2.5 and 2.6 each point of P can be inserted in constant time, so we require $O(n)$ time to construct $\mathrm{DT}(Q \cup P)$. A further $O(n)$ expected time is then needed to obtain $\mathrm{DT}(P)$ using Theorem 1.1. This yields the desired result.

Theorem 2.7 *Let* $\mathcal{R} = \langle R_1, \ldots, R_n \rangle$ *be a sequence of disjoint planar unit disks. In* $O(n \log n)$ *time and using* $O(n)$ *space we can build a scaffolding for* $\mathcal{R}$ *that can be collapsed to a Delaunay triangulation for a precise input in* $O(n)$ *expected time.*

3 Disks of Different Sizes: Quadtree-approach

We now extend Theorem 2.7 to differently-sized disks using a somewhat more involved approach. The main idea is to find a quadtree T such that each cell of T meets a bounded number of regions of $\mathcal{R}$. To process an input P we locate the points of P in T, compute a quadtree T' for P, and then apply Theorem 1.2. With the additional structure of the quadtree we can handle disks of varying sizes.

We define a *free quadtree* T to be an ordered rooted tree that corresponds to a hierarchical decomposition of the plane into closed axis-aligned square *boxes*. Each node v of T has a square box B_v associated to it, according to the following rules:

1. If w is a descendant of v in T, then B_w is contained in B_v.
2. If v and w are not related, then the interiors of B_v and B_w are disjoint.

In other words, the boxes B_v constitute a *laminar family* of squares in $\mathbb{R}^2$. We say the *size* of a node is the side length of its box. With each node v, we associate the *cell* C_v that consists of the part of B_v that is not covered by the children of v.[3] Any two distinct cells C_v and C_w have disjoint interiors, and the union of the cells of all nodes of T covers the root square.

[3]Note that C_v could be disconnected or empty.

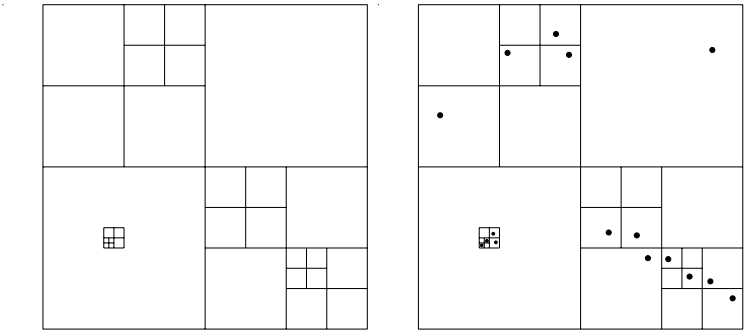

Fig. 2 (**a**) A quadtree. The *lower left box* contains a cluster node. (**b**) The quadtree is a valid quadtree for this set of points

A standard quadtree [20] is a special case of a free quadtree, and in particular has only two types of nodes: *internal nodes* v with exactly four children half the size of v, and *leaf nodes* without any children. In this section we define a (compressed) quadtree as a standard quadtree, but with the addition of a third type of node, which we call *cluster node*: a node v with just one child w, whose size is smaller than its parent by at least a large constant factor 2^c. We require that the horizontal and vertical distances between the boundary B_w and the boundary of B_v are either zero or at least the size of B_w. Figure 2(a) shows an example of a quadtree of this type. Cluster nodes ensure that the complexity of T can be kept linear [8, Sect. 3.2].

Given a planar point set P, we say that T is a quadtree *for* P if the following properties hold:

1. Each leaf v of T contains at most one point of P inside C_v.
2. Other nodes v of T contain no points of P inside C_v.
3. The root box of T contains all points of P.
4. The complexity of T is $O(|P|)$.

Figure 2(b) shows a set of points for which the quadtree is valid.

To apply Theorem 1.2 we need to preprocess $\mathcal{R}$ into a quadtree T such that any cell C_v in T intersects only constantly many disks. While we could consider the disks directly, we will instead use a quadtree T for a point set Q representing the disks. For each disk we include in Q its center and top-, bottom-, left- and rightmost point. Then, T can be constructed in $O(n \log n)$ time (see Appendix A).

Lemma 3.1 *Every cell C_v of T is intersected by $O(1)$ disks in $\mathcal{R}$.*

Proof There are three types of nodes to consider. First, if v is an internal node with four children, then C_v is empty and the condition holds trivially. Next, suppose that v is a leaf node, so $C_v = B_v$. If a disk D intersects B_v and does not contain a corner of B_v, then B_v must either contain D's center or one of its four extreme points [6]. Thus, B_v intersects at most 5 disks, one for each corner and one for the point of Q it contains. See Fig. 3(a) for an example.

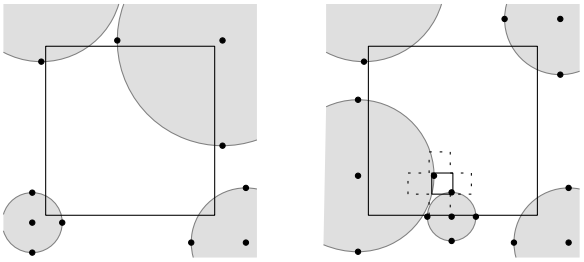

Fig. 3 (**a**) At most 4 disjoint disks can intersect any given box B belonging to a leaf of the quadtree, without one of their points being inside B. (**b**) For a box B belonging to a parent of a cluster node with box C, slightly more disks can intersect the interior of $B \setminus C$, but not more than 4 can cover any of the four neighboring boxes, so a crude upper bound is 20

Now suppose v is a cluster node, with child w. Then $C_v = B_v \setminus B_w$. We know there are no points of Q in C_v, and there are at most four disks that have all their representative points outside B_v. So it remains to count the disks that intersect B_v, do not cover a corner of B_v, and have an extreme point or their center in B_w. For this, consider the at most four orthogonal neighbors of B_w in B_v (i.e., copies of B_w directly to the left, to the right, above and below B_w) that lie inside B_v. Using the same argument as above, each of these neighbors meets at most four disks, and every disk D with an extreme point or center in B_w that intersects C_v also meets one of the orthogonal neighbors (if D has no extreme point or center in an orthogonal neighbor and does not cover any of its corners, it has to cover its center), which implies the claim, because by our assumption about cluster nodes all the orthogonal neighbors are completely contained in B_v. Figure 3(b) shows an example involving a cluster node. □

Theorem 3.2 *Let $\mathcal{R} = \langle R_1, \ldots, R_n \rangle$ be a sequence of disjoint planar disks (not necessarily all of the same size). In $O(n \log n)$ time and using $O(n)$ space we can build a scaffolding for $\mathcal{R}$ that can be collapsed into a Delaunay triangulation for a concrete point set in $O(n)$ expected time.*

Proof We construct Q and the quadtree T for Q as described above. For each R_i we store a list with the leaves in T that intersect it. By Lemma 3.1, the total size of these lists, and hence the complexity of the data structure, is linear. Now we describe how to process an input: let $P = \langle p_1, \ldots, p_n \rangle$ be the input sequence. For each p_i we find the node v of T such that $p_i \in C_v$ by traversing the list for R_i. This takes linear time. Since each leaf of T contains constantly many input points, we can turn T into a quadtree for P in linear time. We now compute $\mathrm{DT}(P)$ via Theorem 1.2.[4] □

4 Overlapping Disks: Deflated Quadtrees

We extend the approach to disks with limited overlap. Now $\mathcal{R}$ contains n planar disks such that no point is covered by more than k disks. The parameter k is called

[4]We are slightly cheating here, because when turning T into a quadtree for P we need to be careful about handling cluster nodes that contain an input point. We will explain this in the next section, where we consider a more general setting.

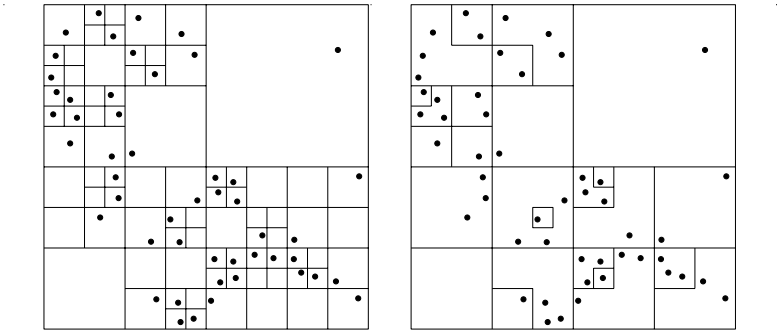

Fig. 4 (**a**) A set of points and a quadtree for it. (**b**) A 3-deflated version of the quadtree

the *depth* of $\mathcal{R}$, and Aronov and Har-Peled [1] showed that k can be approximated up to a constant factor in $O(n \log n)$ time. It is easily seen that finding a Delaunay triangulation for points from these regions takes $\Omega(n \log k)$ time in the worst case in the algebraic decision tree model, and we show that this bound can be achieved.

The general strategy is the same as in Sect. 3. Let Q be the $5n$ representative points for $\mathcal{R}$, and let T be a quadtree for Q. As before, T can be found in time $O(n \log n)$ and has complexity $O(n)$. However, the cells of T can now be intersected by $O(k)$ regions, rather than $O(1)$. Since our data structure stores all the cell-disk incidences, this means that the space requirement would become $O(nk)$, which is too large. However, we can avoid this by reducing the complexity of T until it only has $O(n/k)$ cells, while maintaining the intersection property. Then, we share the more detailed structures between the regions in $\mathcal{R}$, thus saving space. For this we introduce the notion of λ-*deflated* quadtrees.

For a positive integer $\lambda \in \mathbb{N}$, a λ-deflated quadtree T' for a point set Q has the same general structure as the quadtrees from the previous section, but it has lower complexity: each node of T' can contain up to λ points of Q in its cell and there are only $O(n/\lambda)$ nodes. We distinguish four different types of nodes: (i) *leaves* are nodes v without children, with up to λ points in C_v; (ii) *internal nodes* v have four children of half their size covering their parent, and $C_v = \emptyset$; (iii) *cluster nodes* are, as before, nodes v with a single—much smaller—child, and with no points in C_v; (iv) finally, a *deflated node* v has only one child w—possibly much smaller than its parent—and additionally C_v may contain up to λ points. Cluster nodes and deflated nodes are very similar, but they play slightly different roles while collapsing the scaffolding. An example of a quadtree and a 3-deflated version of it is shown in Fig. 4.

Given a quadtree T for Q, a λ-deflated quadtree T' can be found in linear time: for every node v in T, compute $n_v = |B_v \cap Q|$. This takes $O(n)$ time with a postorder traversal. Then, T' is obtained by applying `DeflateTree` to the root of T (Algorithm 1). Since `DeflateTree` performs a simple top-down traversal of T, it takes $O(n)$ time.

Lemma 4.1 *A λ-deflated quadtree T' produced by Algorithm 1 has $O(n/\lambda)$ nodes.*

Algorithm 1 Turning a quadtree into a λ-deflated quadtree

Algorithm `DeflateTree`(v)

1. If $n_v \leq \lambda$, return the tree consisting of v.
2. Let T_v be the subtree rooted in v, and let z be a node in T_v with the smallest value n_z such that $n_z > n_v - \lambda$. Note that z could be v.
3. For all children w of z, let $T'_w = \texttt{DeflateTree}(w)$.
4. Build a tree T'_v by picking v as the root, z as the only child of v, and linking the trees T'_w to z. If $v \neq z$ and C_v is not empty, then v is a deflated node; if $v \neq z$ and C_v is empty, v is a cluster node. Return T'_v as the result.

Proof Let T'' be the subtree of T' that contains all nodes v with $n_v > \lambda$, and suppose that every cluster node in T'' has been contracted with its child. We will show that T'' has $O(n/\lambda)$ nodes, which implies the claim, since no two cluster nodes are adjacent, and because all the non-cluster nodes in T' which are not in T'' must be leaves. We count the nodes in T'' as follows: (i) since the leaves of T'' correspond to disjoint subsets of Q of size at least λ, there are at most n/λ of them; (ii) the bound on the leaves also implies that T'' contains at most n/λ nodes with at least two children; (iii) the number of nodes in T'' with a single child that has at least two children is likewise bounded; (iv) when an internal node v has a single child w that also has only a single child, then by construction v and w together must contain at least λ points in their cells, otherwise they would not have been two separate nodes. Thus, we can charge $\lambda/2$ points from Q to v, and the total number of such nodes is $2n/\lambda$. □

Lemma 4.2 *Let T' be a k-deflated quadtree for Q. Every cell C_v of T' is intersected by $O(k)$ disks of $\mathcal{R}$.*

Proof Treat deflated nodes like cluster nodes and note that the center and corners of every box of T' can be covered by at most k disks. Now the lemma follows from the same arguments as we used in the proof of Lemma 3.1. □

Theorem 4.3 *Let $\mathcal{R} = \langle R_1, \ldots, R_n \rangle$ be a sequence of planar disks such that no point is covered by more than k disks. In $O(n \log n)$ time and using $O(n)$ space we can build a scaffolding for $\mathcal{R}$ that can be collapsed to the Delaunay triangulation for a concrete point set in $O(n \log k)$ expected time.*

Proof It remains to show how to preprocess T' so that it can be collapsed in time $O(n \log k)$. By Lemmas 4.1 and 4.2, the total number of disk-cell incidences in T' is $O(n)$. Thus, in $O(n)$ total time we can find for each $R \in \mathcal{R}$ the list of nodes in T' whose cells it intersects. Next, we determine for each node v in T' the portion X_v of the original quadtree T inside the cell C_v and build a point location data structure for X_v. Since X_v is a partial quadtree for at most k points, it has complexity $O(k)$, and since the X_v are disjoint, the total space requirement and construction time are linear. This finishes the preprocessing.

To process a point set, we first locate the input points P in the cells of T' just as in Theorem 3.2. This takes $O(n)$ time. Then we use the point location structures for the

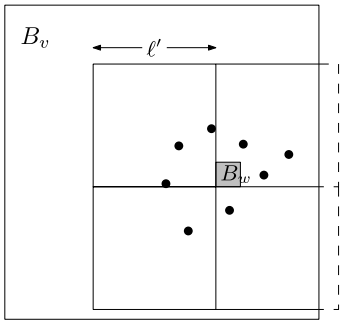

Fig. 5 We align four boxes of side length ℓ' with B_w to obtain a bounding box for P' and B_w. Note that this box may intersect the boundary of B_v

X_v to locate P in T in total time $O(n \log k)$. Finally we turn T into a quadtree for P in time $O(n \log k)$, and find the Delaunay triangulation in time $O(n)$, as before.

We now explain how T is turned into a quadtree for P: for cells corresponding to leaf nodes, we can just use a standard algorithm for computing quadtrees, which takes $O(\alpha \log k)$ time for a cell that contains α points, since $\alpha = O(k)$ by Lemma 4.2. For cells that correspond to cluster nodes, we must work harder in order to avoid the need for the floor function: suppose v is a cluster node with child w, such that C_v contains α points P' from P. We sort P' according to x- and y-coordinates, and then determine a bounding box for B_w and P'. Let ℓ be the side length of this bounding box. Now we find in $O(\log k)$ time an integer $0 \le \gamma \le \alpha$, such that $|B_w| \le 2^{-c\gamma}\ell$ and either $\gamma = \alpha$ or $|B_w| \ge 2^{-c\gamma-1}\ell$, where $|B_w|$ denotes the size of B_w. Then, we set $\ell' = 2^{c\gamma+1}|B_w|$ if $\gamma \ne \alpha$, and $\ell' = \ell$ otherwise. Align four boxes of side length ℓ' with B_w (see Fig. 5) and use the result as the bounding box for the quadtree T'' that contains B_w and P'. This ensures that no edge of T'' intersects B_w, because all non-cluster nodes of T'' have size at least $2^{-c\alpha}\ell'$. If during the construction of T'' the box B_w is again contained in a cluster node, we repeat the procedure (without the sorting step), which takes another $O(\log k)$ time. However, this cluster node will contain strictly less than α points from P' (because it is significantly smaller than the bounding box of T''), so the total cost for the bounding box computations cannot exceed $O(\alpha \log k)$. There is one subtlety: the bounding box of T'' might intersect the boundary of B_v. If this happens, we clip T'' to B_v. The resulting quadtree is *skewed*, which means that its cells can be shifted relative to their parent, and some of them may even be clipped. In Appendix B we argue that Theorem 1.2 still holds in this case. □

5 Realistic Input Models

For unconstrained planar input regions any algorithm for our problem requires time $\Omega(n \log n)$ even after preprocessing. This already happens for disks with arbitrary overlap. In the following we show how to obtain better bounds for *realistic input models* [7]. The results of the previous sections readily generalize, because a point set representing the regions—like the set Q for the disks—often exists in such models, e.g., if the regions are fat. Thus, we immediately get an algorithm for fat regions, for which we also provide a matching lower bound. We then demonstrate how to handle

situations where a set like Q cannot be easily constructed by presenting an algorithm for *thick regions*. While fatness and thickness characterize individual regions, there are also models for sets of regions, in particular *low density*, for which the same algorithm as for fat regions applies.

5.1 Models

In this section, we consider the following notions of realistic input regions.

- *Fatness*. Let β be a constant with $0 < \beta \leq 1$. A region $R \subset \mathbb{R}^d$ is β-*fat* if for any d-ball D with center in R and with $\partial D \cap \partial R \neq \emptyset$, we have $\text{volume}(R \cap D) \geq \beta \cdot \text{volume}(D)$, where $\text{volume}(\cdot)$ denotes the d-dimensional volume.
- *Thickness*. A region $R \subset \mathbb{R}^d$ is β-*thick* if $\text{volume}(R) \geq \beta \cdot \text{volume}(D_{\min}(R))$, where $D_{\min}$ denotes the smallest d-ball enclosing R.
- *Density*. Let $\lambda \geq 1$. A set of regions $\mathcal{R}$ in $\mathbb{R}^d$ is λ-*dense* if any d-ball D intersects at most λ regions $R \in \mathcal{R}$ with $\text{diam}(R) \geq \text{diam}(D)$, where $\text{diam}(\cdot)$ denotes the diameter.

There is a close connection between fatness and density [32]:

Lemma 5.1 *Let $\mathcal{R}$ be a set of β-fat regions in $\mathbb{R}^d$ such that no point is covered by more than k regions. Then $\mathcal{R}$ is $(2^d k/\beta)$-dense.*

Proof This is a direct generalization of the proof of Theorem 3.1 in [7]: given a d-ball D with radius r, consider the d-ball D' with the same center and radius $2r$. Let R be a β-fat region with $\text{diam}(R) \geq r$ that intersects D without covering it completely. We can place a d-ball D'' of radius r in D' with center in R and intersecting ∂R. By fatness, R covers a β-fraction of D'' and therefore a $(\beta/2^d)$-fraction of D'. There can be at most $2^d k/\beta$ such regions and therefore at most $2^d k/\beta$ regions intersect D, as required for low density. □

Strong guarding sets. We will use the fact that λ-dense sets of regions can be *guarded*. Let $\kappa \in \mathbb{N}$, and $\mathcal{R}$ be a set of regions in $\mathbb{R}^d$. A point set $Q \subseteq \mathbb{R}^d$ is called a κ-*guarding set* (*against axis-aligned d-cubes*) for $\mathcal{R}$, if any axis-aligned d-cube not containing a point from Q intersects at most κ regions from $\mathcal{R}$. For instance, the point set Q from the previous sections is a 4-guarding set for disjoint disks [6]. It is also a $4k$-guarding set for disks which do not cover any point more than k times.

One can show that if an axis-aligned d-cube contains m points of Q, then it intersects $O(2^d \kappa m)$ regions in $\mathcal{R}$ [6, Theorem 2.8]. If $\mathcal{R}$ is λ-dense, this bound can be improved as follows: assume each point in Q is assigned to a region in $\mathcal{R}$. We call Q a κ-*strong-guarding set* for $\mathcal{R}$ if any axis-aligned d-cube containing m points of Q intersects at most κ regions plus the regions assigned to the m points. This definition is motivated by the following relation between density and guardability.

Lemma 5.2 *For a λ-dense set of regions $\mathcal{R}$ in $\mathbb{R}^d$, the corners of the bounding boxes of $\mathcal{R}$ constitute a $(\lceil\sqrt{d}\rceil^d \lambda)$-strong-guarding set* (*with corners assigned to the corresponding region*).

Proof Let $Q(\mathcal{R})$ denote the corners of the bounding boxes of $\mathcal{R}$, and let B be an axis-aligned d-cube. Consider the largest subset $\mathcal{R}' \subseteq \mathcal{R}$ such that $Q(\mathcal{R}') \cap B = \emptyset$. Since $\mathcal{R}'$ is still λ-dense, we can use the known relation between density and guardability, namely, that a λ-dense set of regions $\mathcal{R}'$ is $(\lceil\sqrt{d}\rceil^d\lambda)$-guarded by $Q(\mathcal{R}')$ [4, Lemma 2.8], [6, Theorem 3.3]. Thus, B meets at most $\lceil\sqrt{d}\rceil^d\lambda$ regions in $\mathcal{R}'$, as claimed. □

5.2 Building and Collapsing the Scaffolding for Realistic Input

Since the argument in the proof of Lemma 3.1 (and of Lemma 4.2) is based on axis-aligned boxes, it directly generalizes to the quadtree for a guarding set.

Lemma 5.3 *Let $\mathcal{R}$ be a set of planar regions and Q a κ-strong-guarding set for $\mathcal{R}$. Let T' be a κ-deflated quadtree of Q. Then any cell of T intersects $O(\kappa)$ regions.* □

We say that $\mathcal{R}$ is *traceable* if we can find the m incidences between the n regions in $\mathcal{R}$ and the l cells of a deflated quadtree T in $O(l+m+n)$ time. For example, this holds for polygonal regions of total complexity $O(|\mathcal{R}|)$.

Theorem 5.4 *Let $\mathcal{R} = \langle R_1, \ldots, R_n\rangle$ be a sequence of traceable planar regions with linear-size κ-strong-guarding set Q, where κ is not necessarily known, but Q is. In $O(n \log n)$ time and using $O(n)$ space we can preprocess $\mathcal{R}$ into a scaffolding that can be collapsed to a concrete point set in $O(n \log \kappa)$ expected time.*

Proof If κ were known, we could construct a κ-deflated quadtree for Q, and the theorem would follow directly from Lemma 5.3 and the techniques from Sect. 4. Fortunately, even if we do not know κ, we can find a suitable λ-deflated tree with $\lambda \in O(\kappa)$ by an exponential search on λ, i.e., for $\lambda = 2, 4, 8, \ldots$ we build a λ-deflated tree and trace the regions of $\mathcal{R}$ in the tree. We abort and continue with the next λ if there is a box that intersects more than $c\lambda$ regions for a constant $c \geq 6$, or if the tracing process takes too long. Otherwise, we have found a suitable deflated quadtree. Since it takes linear time to compute a deflated quadtree from a quadtree, this search needs $O(n \log n)$ time. □

Corollary 5.5 *Let $\mathcal{R} = \langle R_1, \ldots, R_n\rangle$ be a sequence of planar traceable λ-dense regions In $O(n \log n)$ time and using $O(n)$ space we can build a scaffolding for $\mathcal{R}$ that can be collapsed to a Delaunay triangulation for a concrete point set in $O(n \log \lambda)$ expected time.*

Proof Combine Lemma 5.2 with Theorem 5.4. □

Corollary 5.6 *Let $\mathcal{R} = \langle R_1, \ldots, R_n\rangle$ be a sequence of planar traceable β-fat regions such that no point is covered by more than k regions. In $O(n \log n)$ time and using $O(n)$ space we can build a scaffolding for $\mathcal{R}$ that can be collapsed to a Delaunay triangulation for a concrete point set in $O(n \log(k/\beta))$ expected time.*

Proof This follows from Lemma 5.1 and Corollary 5.5. □

Finally, we consider β-thick regions. Although thickness does not give us a guarding set, we can still obtain results for a set of regions $\mathcal{R}$ if the ratio between the largest and smallest region in $\mathcal{R}$ is bounded (we measure the size by the radius of the smallest enclosing circle).

Theorem 5.7 *Let $\mathcal{R}$ be a sequence of n β-thick k-overlapping regions such that the ratio of the largest and the smallest region in $\mathcal{R}$ is r. In $O(n \log n)$ time we can preprocess $\mathcal{R}$ into a linear-space scaffolding that can be collapsed to the Delaunay triangulation of a concrete point set in time $O(n(\log(k/\beta) + \log\log r))$.*

Proof Subdivide the regions into $\log r$ groups such that in each group the radii of the minimum enclosing circles differ by at most a factor of 2. For each group $\mathcal{R}_i$, let ρ_i be the largest radius of a minimum enclosing circle for a region in $\mathcal{R}_i$. We replace every region in $\mathcal{R}_i$ by a disk of radius ρ_i that contains it. This set of disks is at most $(2k/\beta)$-overlapping, so we can build a data structure for $\mathcal{R}_i$ in $O(n_i \log n_i)$ time by Theorem 4.3. To process an input, we handle each group in $O(n_i \log(k/\beta))$ time and then use Kirkpatrick's algorithm [25] to combine the triangulations in time $O(n \log\log r)$. □

5.3 Lower Bounds

We shall now see that most of the results from the previous section cannot be improved. First, we show that the $O(n \log(1/\beta))$ bound for disjoint fat regions from Corollary 5.6 is optimal.

Theorem 5.8 *For any n and $b \in \{2, \ldots, n\}$, there exists a set $\mathcal{R}$ of n planar disjoint $\Theta(b^{-2})$-fat rectangles such that it takes $\Omega(n \log b)$ time to find a Delaunay triangulation for a point set with exactly one point from each region in $\mathcal{R}$ in the algebraic computation tree model.*

Proof We adapt a lower bound by Djidjev and Lingas [18, Sect. 4]. For this we consider the problem $(b, 1)$-CLOSENESS: given $k = n/b$ sequences $\mathbf{x}_1, \ldots, \mathbf{x}_k$, each containing b real numbers in $[0, 2b]$, we need to decide whether any $\mathbf{x}_i$ contains two numbers with difference at most 1. To see that any algebraic decision tree for $(b, 1)$-CLOSENESS has depth $\Omega(n \log b)$, let $W' \subseteq \mathbb{R}^n$ be defined as

$$W' = \left\{(\mathbf{x}_1, \ldots, \mathbf{x}_k) \mid |x_{ij} - x_{il}| > 1 \text{ for } 1 \le i \le k; 1 \le j < l \le b\right\},$$

where x_{ij} is the jth coordinate of $\mathbf{x}_i$. Let $W = W' \cap [0, 2b]^n$. Since W has at least $(b!)^{n/b}$ connected components, Ben-Or's lower bound [3, Theorem 5] implies that we need depth at least $\Omega((n/b)\log(b!)) = \Omega(n \log b)$, as claimed.

Now, we construct $\mathcal{R}$. Let $\varepsilon = 1/(3b)$ and consider the b intervals on the x-axis given by $B_\varepsilon[1/b]$, $B_\varepsilon[2/b], \ldots, B_\varepsilon[1]$, where $B_\varepsilon[x]$ denotes the one-dimensional closed ε-ball around x. Extend the intervals into $\Theta(b^{-2})$-fat rectangles with side lengths 2ε and $2b$. These rectangles constitute a *group*; see Fig. 6. Now, $\mathcal{R}$ consists

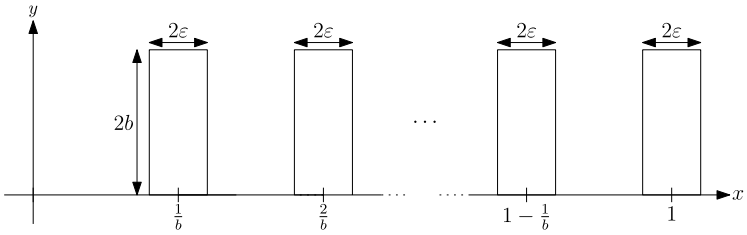

Fig. 6 A group consists of b rectangles of fatness $\Theta(b^{-2})$

of k congruent groups $G_1, \ldots, G_k$, translated in y-direction so that they are sufficiently far away from each other. Let $\mathbf{x}_1, \ldots, \mathbf{x}_k$ be an instance of $(b, 1)$-CLOSENESS. The input P consists of k sets $P_1, \ldots, P_k$, one for each $\mathbf{x}_i$. Each P_i contains b points, one from every rectangle in G_i: $P_i = \langle (1/b, x_{i1}), (2/b, x_{i2}), \ldots, (1, x_{ib}) \rangle + (0, y_i)$, where $(0, y_i)$ denotes the translation vector for G_i. Clearly, P can be computed in $O(n)$ time. The argument by Djidjev and Lingas [18, Lemma 2] shows that if the Voronoi cells for P_i do not intersect the y-axis according to the sorted order of $\mathbf{x}_i$, then $\mathbf{x}_i$ contains two numbers with difference less than 1. Thus, by examining the intersections between the y-axis and the Voronoi cells (which can be found in linear time given $\mathrm{DT}(P)$), we can decide $(b, 1)$-CLOSENESS in linear time from $\mathrm{DT}(P)$: either they represent the sorted order of each $\mathbf{x}_i$, in which case the answer is easily determined; or they do not, in which case the answer is yes. □

Second, we report an example that was shown to us by Mark de Berg and that indicates that the factor of $\log\log r$ in Theorem 5.7 cannot be improved.

Theorem 5.9 *For any n there exists a set $\mathcal{R}$ of n disjoint 0.1-thick regions such that it takes $\Omega(n \log\log r)$ time to find the Delaunay triangulations for a point set with exactly one point from each region in $\mathcal{R}$, in the algebraic computation tree model. Here, r is the ratio between the largest and smallest region in $\mathcal{R}$.*

Proof Consider the situation in Fig. 7. For $k = 1, \ldots, n$, region R_i consists of a $2^k \times 2^k$ square extended by a line segment of length 2^k. Clearly, all the regions R_i are 0.1-thick, and the ratio r is $\Omega(2^n)$. Now, if the points in the input all lie in the region bounded by the dashed rectangle, we only know that the inputs are contained in equally spaced line segments. For this case, the lower bound by Djidjev and Lingas [18] that we countered in the proof of Theorem 5.8 implies that an input needs $\Omega(n \log n) = \Omega(n \log\log r)$ time in the worst case to be processed, as claimed. □

6 Higher Dimensions

Finally, we discuss how our results generalize to higher dimensions. Of course we cannot expect linear worst-case processing time, since in general the complexity of a d-dimensional Delaunay triangulation might be $\Omega(n^{\lceil d/2 \rceil})$. However, in many "typical" situations this worst-case behavior does not occur, and we therefore express the

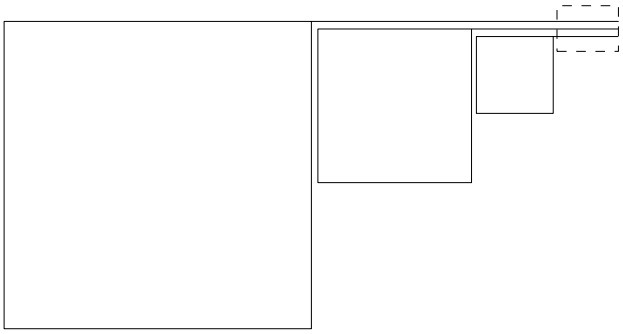

Fig. 7 A lower bound for thick regions

processing time in terms of the expected structural change $C(P)$ of a randomized incremental construction of $\mathrm{DT}(P)$. The generalization of Theorem 1.2 to higher dimensions says that $\mathrm{DT}(P)$ can be computed from the quadtree in time $O(C(P))$. Again, we need to extend the theorem to skewed quadtrees, which works just like in the planar case (Appendix B). Now, all of the realistic input models above are defined for any dimension. Consider a sequence of n traceable regions $\mathcal{R}$ in any fixed dimension with linear-size κ-strong-guarding set Q. We can find a κ-deflated quadtree T for Q in $O(n \log n)$ time. To process an input we can again use T to obtain a skewed quadtree for the points in $O(n \log \kappa)$ time. Adding the time for constructing the Delaunay triangulation from the quadtree, the total expected time for a point set P is $O(n \log \kappa + C(P))$. Thus, while the techniques generalize to higher dimensions, the approach is only useful if $C(P)$ is expected to be linear or near-linear.

7 Conclusions

We give an alternative proof of the result by Löffler and Snoeyink [29] with a much simpler, albeit randomized, algorithm that avoids heavy machinery. For our simplified approach, we need randomization only when we apply Theorem 1.1 to remove the scaffold, and finding a deterministic algorithm for hereditary Delaunay triangulations remains an intriguing open problem.

Using quadtrees, we also obtain optimal results for overlapping disks of different sizes and fat regions. Furthermore, we are able to leverage known facts about guarding sets to handle many other realistic input models. Our techniques seem quite specific to Delaunay triangulations, and it is an interesting question whether similar results can be proven for other geometric structures.

Acknowledgements The results in Sect. 2 were obtained at the NICTA Workshop on Computational Geometry for Imprecise Data, December 18–22, 2008 at the NICTA University of Sydney Campus. We would like to thank the other workshop participants, namely Hee-Kap Ahn, Sang Won Bae, Dan Chen, Otfried Cheong, Joachim Gudmundsson, Allan Jørgensen, Stefan Langerman, Marc Scherfenberg, Michiel Smid, Tasos Viglas and Thomas Wolle, for helpful discussions and for providing a stimulating working environment. We would also like to thank David Eppstein for answering our questions about [8] and pointing us to [9]. We would like to thank Mark de Berg for discussing realistic input models with us and

for providing Theorem 5.9. Finally, we would like to thank an anonymous reviewer for valuable comments that helped improve the readability of the paper.

Appendix A: Computing Compressed Quadtrees in $O(n \log n)$ Time

It is well known that given a planar n-point set P, we can compute a compressed quadtree T for P in $O(n \log n)$ time. However, the details of the construction are a bit involved [19], and for the reader's convenience we describe here a possible implementation that achieves the claimed running time on a pointer machine. The basic algorithm is as follows: sort the points in x- and y-direction, find a bounding box for P, and repeatedly split the boxes in the obvious way. If after c splits of the current box the size of the point set P' inside it has not decreased, we find a bounding box for P', create a cluster node and recursively compute a quadtree for P'.

In order to perform the splitting efficiently, we need to maintain the x- and y-orderings of the points contained in the current box. To do this, we do the splitting in two steps: first in x-direction, then in y-direction. To split in x-direction, we traverse the x-list simultaneously from both ends to find the split sets in time proportional to the size of the smaller set. Let P_1 and P_2 be the two resulting sets, and let n_1 and n_2 be their sizes, with $n_1 \leq n_2$. Using appropriate pointers, we find the points of P_1 in the y-list and remove them. Now we need to create the sorted y-list for P_1: if $n_1 \leq n_2^{1/2}$, we just sort P_1 according to y-coordinate in $O(n_1 \log n_1)$ time. Otherwise, we use a pointer-based radix sort for P_1 that takes $O(n_1)$ time. For this, we need to maintain an appropriate data structure with each of the y-lists.[5] This data structure is created when a y-list is split off and updated every time the size of a y-list halves, which leads to a linear time overhead. The total time needed for the splitting in x-direction obeys the recursion

$$T(n) = \begin{cases} T(n_1) + T(n_2) + O(n_1 \log n_1), & \text{if } n_1 \leq n_2^{1/2}, \\ T(n_1) + T(n_2) + O(n_1), & \text{if } n_2^{1/2} \leq n_1 \leq n_2. \end{cases}$$

This solves to $T(n) = O(n \log n)$. The splitting in y-direction is done in the same way, and the total time needed for the construction of the quadtree is $O(n \log n)$, as claimed.

Appendix B: From Quadtrees to Delaunay Triangulations

Theorem 1.2 is proven in [11]. However, because of a technicality that is described in Sect. 4, we need a slight generalization of the theorem. Therefore, here we first

[5] That is, we need a pointer structure that allows us to associate three-digit integers with the points according to their position in the y-list.

briefly sketch the structure of that proof, and then show how it can be adapted to our needs. The proof is based on a chain of reductions from Delaunay triangulations to nearest-neighbor graphs to well-separated pair decompositions to quadtrees.

The authors show that to compute $\mathrm{DT}(P)$, it suffices to find the nearest neighbor graphs $\mathrm{NN}(P_1), \ldots, \mathrm{NN}(P_t)$ for each level of an appropriate gradation $\emptyset = P_0 \subseteq P_1 \subseteq \cdots \subseteq P_t = P$ with $\sum_{i=0}^{t} |P_i| = O(n)$. By a classic reduction [12], $\mathrm{NN}(P_1), \ldots, \mathrm{NN}(P_t)$ can be found in linear time, once we know ε-well-separated pair decompositions (ε-WSPDs) for $P_1, \ldots, P_t$ of linear size and for appropriate ε. These ε-WSPDs can in turn be derived from compressed quadtrees for these sets. By assumption of the theorem, we have a quadtree T for $P = P_t$, and by successively pruning T, we can get quadtrees for $P_1, \ldots, P_{t-1}$ in linear time.

Now we come to the adaptation of the proof. In Sect. 4, we defined a *skewed* quadtree as a standard compressed quadtree, but one where clusters can be shifted relative to their parents and parts of the cluster cells might be clipped. We need to prove that Theorem 1.2 still holds in the case where we are given a skewed quadtree for P, rather than a regular quadtree. Note that all steps outlined above can go through unchanged, except that now we need to go from skewed quadtrees to ε-WSPDs. For this, we take the algorithm that can compute an ε-WSPD based on a standard compressed quadtree [12, 13] and check that it still works if the quadtree is skewed. We make a key observation about skewed quadtrees first.

Observation B.1 *Let v be a node of a skewed quadtree T, and let d be the size its box would have without clipping. Then there is a box B adjacent to B_v (possibly diagonally) such that the volume of B is at least cd^2 for some constant c.*

Also, recall the definition of an ε-WSPD for a point set P. It is a collection of pairs $\{(P_1, Q_1), \ldots, (P_m, Q_m)\}$ with $P_i, Q_i \subseteq P$ such that each pair (P_i, Q_i) is *ε-well-separated*, which means that the diameters of P_i and Q_i are both at most ε times the minimum distance between P_i and Q_i. Furthermore, we require that every pair $(p, q) \in P \times P$ of distinct points is in $P_i \times Q_i$ or $Q_i \times P_i$ for exactly one i. We call m the *size* of the ε-WSPD.

Now we are ready to follow the argument. Let T be a skewed quadtree for P, and let us see how to find a well-separated pair decomposition for P of linear size in time $O(|P|)$, given T. Our presentation follows Chan [13], with some small changes to adapt the argument to skewed quadtrees. The WSPD is computed by performing the function $\texttt{wspd}(v)$ on the root v of T. Refer to Algorithm 2.

Here, P_v denotes the points contained in B_v, the box corresponding to v, and $|B_v|$ is the size of B_v. Clearly, `wspd` computes an ε-WSPD for P. We need to argue that it has linear size and that the computation takes linear time. For this, it suffices to count the number of calls to $\texttt{wspd}(v_1, v_2)$ such that B_{v_1} and B_{v_2} are not ε-well-separated. By induction, we see that whenever $\texttt{wspd}(v_1, v_2)$ is called, the size of the box corresponding to the parent of v_1 is at least $|B_{v_2}|$ and the size of the parent box for v_2 is at least $|B_{v_1}|$. Without loss of generality, we assume $|B_{v_1}| \leq |B_{v_2}|$, and let r be the parent of v_1. As we noted above, we have $|B_r| \geq |B_{v_2}|$, and by successively subdividing B_r in a quadtree fashion (and shifting if necessary), we obtain a box B such that $B_{v_1} \subseteq B \subseteq B_r$, and such that $|B_{v_2}|/2 \leq |B| \leq |B_{v_2}|$. Since B and B_{v_2} are

Algorithm 2 Finding a well-separated pair decomposition

`wspd`(v)

1. If v is a leaf, return $\emptyset$.
2. Return the union of `wspd`(r) and `wspd`(r_1, r_2) for all children r and pairs of distinct children r_1, r_2 of v.

`wspd`(v_1, v_2)

1. If B_{v_1} and B_{v_2} are ε-well-separated, return (P_{v_1}, P_{v_2}).
2. Otherwise, if $|B_{v_1}| \leq |B_{v_2}|$, return the union of `wspd`(v_1, r) for all children r of v_2.
3. Otherwise, return the union of `wspd`(r, v_2) for all children r of v_1.

not ε-well-separated, B must be within distance $O(|B|/\varepsilon)$ of B_{v_2}. To see that there are only constantly many such boxes, we can use a volume argument, but we need to be careful about boxes which are clipped because they are contained in a cluster that is shifted relative to its parent. However, by Observation B.1, every clipped box has an adjacent box which is only cut by a constant fraction c (if at all). Therefore, the number of boxes that are clipped too much can be at most 8 times the number of boxes that are clipped by at most c, so the total number of nearby boxes is still constant. Hence, the total size of the WSPD is linear.

References

1. Aronov, B., Har-Peled, S.: On approximating the depth and related problems. SIAM J. Comput. **38**(3), 899–921 (2008)
2. Bandyopadhyay, D., Snoeyink, J.: Almost-Delaunay simplices: nearest neighbor relations for imprecise points. In: Proc. 15th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA), pp. 403–412 (2004)
3. Ben-Or, M.: Lower bounds for algebraic computation trees. In: Proc. 16th Annu. ACM Sympos. Theory Comput. (STOC), pp. 80–86 (1983)
4. de Berg, M.: Linear size binary space partitions for uncluttered scenes. Algorithmica **28**(3), 353–366 (2000)
5. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer, Berlin (2008)
6. de Berg, M., David, H., Katz, M.J., Overmars, M.H., van der Stappen, A.F., Vleugels, J.: Guarding scenes against invasive hypercubes. Comput. Geom. Theory Appl. **26**(2), 99–117 (2003)
7. de Berg, M., van der Stappen, A.F., Vleugels, J., Katz, M.J.: Realistic input models for geometric algorithms. Algorithmica **34**(1), 81–97 (2002)
8. Bern, M., Eppstein, D., Gilbert, J.: Provably good mesh generation. J. Comput. Syst. Sci. **48**(3), 384–409 (1994)
9. Bern, M., Eppstein, D., Teng, S.H.: Parallel construction of quadtrees and quality triangulations. Int. J. Comput. Geom. Appl. **9**(6), 517–532 (1999)
10. Bruce, R., Hoffmann, M., Krizanc, D., Raman, R.: Efficient update strategies for geometric computing with uncertainty. Theory Comput. Syst. **38**(4), 411–423 (2005)
11. Buchin, K., Mulzer, W.: Delaunay triangulations in $O(\texttt{sort}(n))$ time and more. In: Proc. 50th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS), pp. 139–148 (2009)
12. Callahan, P.B., Kosaraju, S.R.: A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. J. ACM **42**(1), 67–90 (1995)

13. Chan, T.M.: Well-separated pair decomposition in linear time? Inform. Process. Lett. **107**(5), 138–141 (2008)
14. Chazelle, B.: Triangulating a simple polygon in linear time. Discrete Comput. Geom. **6**, 485–524 (1991)
15. Chazelle, B., Devillers, O., Hurtado, F., Mora, M., Sacristán, V., Teillaud, M.: Splitting a Delaunay triangulation in linear time. Algorithmica **34**(1), 39–46 (2002)
16. Chazelle, B., Mulzer, W.: Computing hereditary convex structures. In: Proc. 25th Annu. ACM Sympos. Comput. Geom. (SoCG), pp. 61–70 (2009)
17. Clarkson, K.L., Seshadhri, C.: Self-improving algorithms for Delaunay triangulations. In: Proc. 24th Annu. ACM Sympos. Comput. Geom. (SoCG), pp. 148–155 (2008)
18. Djidjev, H.N., Lingas, A.: On computing Voronoi diagrams for sorted point sets. Int. J. Comput. Geom. Appl. **5**(3), 327–337 (1995)
19. Eppstein, D.: Approximating the minimum weight Steiner triangulation. Discrete Comput. Geom. **11**(2), 163–191 (1994)
20. Finkel, R.A., Bentley, J.L.: Quad trees: a data structure for retrieval on composite keys. Acta Inform. **4**(1), 1–9 (1974)
21. Franciosa, P.G., Gaibisso, C., Gambosi, G., Talamo, M.: A convex hull algorithm for points with approximately known positions. Int. J. Comput. Geom. Appl. **4**(2), 153–163 (1994)
22. Guibas, L.J., Salesin, D., Stolfi, J.: Epsilon geometry: building robust algorithms from imprecise computations. In: Proc. 5th Annu. ACM Sympos. Comput. Geom. (SoCG), pp. 208–217 (1989)
23. Guibas, L.J., Salesin, D., Stolfi, J.: Constructing strongly convex approximate hulls with inaccurate primitives. Algorithmica **9**, 534–560 (1993)
24. Held, M., Mitchell, J.S.B.: Triangulating input-constrained planar point sets. Inform. Process. Lett. **109**(1), 54–56 (2008)
25. Kirkpatrick, D.G.: Efficient computation of continuous skeletons. In: Proc. 20th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS), pp. 18–27 (1979)
26. van Kreveld, M., Löffler, M.: Largest bounding box, smallest diameter, and related problems on imprecise points. Comput. Geom. Theory Appl. **43**(4), 419–433 (2010)
27. van Kreveld, M.J., Löffler, M., Mitchell, J.S.B.: Preprocessing imprecise points and splitting triangulations. In: Proc. 19th Annu. Internat. Sympos. Algorithms Comput. (ISAAC), pp. 544–555 (2008)
28. Kruger, H.: Basic measures for imprecise point sets in $\mathbb{R}^d$. Master's thesis, Utrecht University (2008)
29. Löffler, M., Snoeyink, J.: Delaunay triangulation of imprecise points in linear time after preprocessing. Comput. Geom. Theory Appl. **43**(3), 234–242 (2010)
30. Nagai, T., Tokura, N.: Tight error bounds of geometric problems on convex objects with imprecise coordinates. In: Jap. Conf. on Discrete and Comput. Geom. LNCS, vol. 2098, pp. 252–263. Springer, Berlin (2000)
31. Ostrovsky-Berman, Y., Joskowicz, L.: Uncertainty envelopes. In: Proc. 21st European Workshop Comput. Geom. (EWCG), pp. 175–178 (2005)
32. van der Stappen, A.F.: Motion planning amidst fat obstacles. Ph.D. thesis, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands (1994)